COMP 4432

Machine Learning

Group Project

Analysis of A Million News Headlines Dataset

Yu Ngo Ting Oscar 20048665D

Leung Kit Chuen 20051248D

Total Page: 10

Table of Content

Chapter 1 Introduction

1.1Background

The rapid spread of information through various online platforms has become a double-edged sword in the digital age. While it enables instant access to news worldwide, it also facilitates the proliferation of fake news, which can significantly distort public perception and lead to misinformation. Fake news, characterized by false information presented as accurate and spread, often to mislead, has become a global concern. Its impact ranges from influencing public opinion on political matters to causing panic during crises like the COVID-19 pandemic. The challenge of distinguishing between genuine and fake news has grown exponentially, necessitating innovative solutions to combat its spread.

Machine learning (ML) provides a promising avenue for addressing the problem of fake news. By leveraging ML algorithms, it is possible to analyze patterns in text data to identify characteristics that differentiate genuine news from fake news. This approach involves training ML models on a dataset of news articles labeled as true or false to learn the distinguishing features of fake news. The effectiveness of these models depends on the quality and diversity of the dataset used for training, as well as the choice of ML algorithms applied.

In this project, our group selected the dataset from the reputable Australian news source ABC (Australian Broadcasting Corporation), providing a comprehensive view of news headlines published over nineteen years. This dataset, formatted in CSV and containing detailed information such as the publish date and headline text, is a rich resource for training ML models to classify news articles as genuine or fake. The dataset's focus on international news, with a volume of two hundred daily articles, ensures broad coverage of significant events and issues, making it ideal for developing a robust fake news classifier.

1.2 Project Objective

As for the project goal, we leverage the power of machine learning to analyze and understand the evolution of news narratives over time. By examining the keywords and themes in the dataset, it is possible to gain insights into the significant events and trends that have shaped the global landscape over the past decade. This project aims to contribute

to the ongoing efforts to combat the spread of fake news by developing a machine-learning model capable of accurately classifying news articles based on their content.

In summary, the background of this project is rooted in recognizing the critical role of accurate information dissemination in society and the potential of machine learning to enhance our ability to discern between genuine and fake news. Through developing and applying a machine learning model trained on a comprehensive dataset of news articles, this project seeks to contribute to the broader efforts to mitigate the impact of fake news on public discourse and decision-making.

1.3 Scope of Study

- Leverage the ABC News dataset as our project root by processing data and implementing labeling and vectorization techniques.
- To select appropriate machine learning algorithms for classification.
- To improve models' performance using various techniques.
- To develop machine learning models that classify news articles as genuine or fake based on their headline text.
- To handle imbalanced datasets, the number of fake news articles may be lower than genuine ones.
- To deal with the potential overfitting issues.

Chapter 2 Model Design and Implementation

Regarding our model design, we have separated the implementation into several parts, namely the initial phase, the tuning phase, and the final phase. In the initial phase, we must prepare the dataset for model training. In addition, we have designed six simple models to observe each of their model performance in classifying the news articles. Also, we would create a specific function to handle the imbalanced dataset issues in this phase and apply the balanced dataset to each initial model. In the tuning phase, we would select two initial models as our target to improve their accuracy and performance. In this stage, we would adopt different techniques to optimize their performance. At the final stage, we would maximize our designed model performance by combining the strengths of multiple models using ensemble methods. In this chapter, we would like to demonstrate several crucial steps in our model implementation process, including data preparation, feature extraction, model training, and advanced techniques for model improvement.

2.1 Data Preparation & Preprocessing

Our group has loaded two datasets, one containing actual news headlines (Headlines) and another with fake news headlines. Only the headline_text column is selected to focus on the text content.

To ensure the uniqueness of headlines, our group has removed the duplicated records from these two datasets, and we believe that this could help reduce the redundancy of the information.

```
#Removing duplicated headlines
Headlines = Headlines.drop_duplicates('headline_text')
Headlines1 = Headlines1.drop_duplicates('title')
```

✓ 0.4s

We must label news authenticity to conduct supervised learning. By labeling each headline as either real or fake, we have provided the machine learning model with "0" (True news) and "1" (False news) as an explicit target variable to learn from. This labeling process is crucial for training models to distinguish between real and fake news based on the textual content of the headlines.

```
#Creating lable for datasets
#million-headlines dataset will be used as real headlines
#fake-and-real-news-dataset & fake-news dataset will be used as fake headlines
Headlines['fake'] = 0
Headlines1['fake'] = 1

✓ 0.0s
```

Then, we combined these two datasets as the target dataset for model training. Overall, the training dataset has 50000 actual headlines and 11698 fake headlines. However, the dataset is imbalanced, with more real headlines than fake ones. As a result, we have to deal with these issues in the after stage. As a result, we have to address this issue at a later stage.

```
#Downsize million-headlines dataset to first 50K rows
data = pd.concat([Headlines[:50000],Headlines1])
print('Training dataset contains: {} Real headlines and {} Fake headlines.'.format(50000,len(Headlines1)))
data.to_csv('Combined_headlines.csv', index=False)
✓ 0.6s

Training dataset contains: 50000 Real headlines and 11698 Fake headlines.
```

In the data preprocessing stage of our text analysis, we undertake several critical steps to refine the headlines dataset. First, we remove stopwords. Stopwords are a collection of common words devoid of significant meaning, such as "the," "is," and "and". By doing so, it can minimize noise and enhance the relevance of the text.

```
import gensim
import nltk as nl
nl.download("stopwords", download_dir='./nltk_data')
nl.download("punkt", download_dir='./nltk_data')
from sklearn.feature_extraction import text


nltk_stopwords = nl.corpus.stopwords.words('english')
gensim_stopwords = gensim.parsing.preprocessing.STOPWORDS
sklearn_stopwords = text.ENGLISH_STOP_WORDS
combined_stopwords = sklearn_stopwords.union(nltk_stopwords,gensim_stopwords)
✓ 6.6s

[nltk_data] Downloading package stopwords to ./nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to ./nltk_data...
[nltk_data]   Package punkt is already up-to-date!


💡 Click here to ask Blackbox to help you code faster
print('NLTK has {} stop words'.format(len(nltk_stopwords)))
print('Gensim has {} stop words'.format(len(gensim_stopwords)))
print('Sklearn has {} stop words'.format(len(sklearn_stopwords)))
print('Combined stopwords list has {} stop words'.format(len(combined_stopwords)))
✓ 0.0s

NLTK has 179 stop words
Gensim has 337 stop words
Sklearn has 318 stop words
Combined stopwords list has 390 stop words
```

Next, we employ the Porter Stemming algorithm to reduce words to their root form, such as transforming "running" into "run." This technique aids in grouping similar words, simplifying the text, and making it easier for the machine-learning model to identify patterns.

```
from nltk.stem import PorterStemmer
porter_stemmer = PorterStemmer()
✓ 0.0s
```

Following this, we apply lowercasing to each headline. We converted it to lowercase and split it into individual words. This process standardizes the text by making it case-insensitive and prepares it for subsequent analysis by breaking it into manageable units. Lastly, we filter out any remaining stop words from the headlines, a crucial step that further reduces noise and focuses on the most meaningful words, enhancing the quality and efficiency of our text data for analysis.

```
data['headline_text'] = data['headline_text'].apply(lambda x: x.lower())
data['headline_text'] = data['headline_text'].apply(lambda x: ' '.join([word for word in x.split() if word.isalpha()]))
data['headline_text'] = data['headline_text'].apply(lambda x: ' '.join([porter_stemmer.stem(word) for word in x.split()]))
data['headline_text'] = data['headline_text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (combined_stopwords)]))
```
✓ 12.7s

2.2 Text vectorization

Vectorization assists the machines in understanding the textual contents by converting them into meaningful numerical representations. Before training models, we adopted Term Frequency-Inverse Document Frequency (TF-IDF) as the vectorization technique to assign a weight to each word of the headline text from our training dataset. In the code below, we have initialized the "TfidfVectorizer" to tokenize the input text into individual words and only considered the 300 most frequent words. We set up various classifiers for later model training usage. This is crucial because TF-IDF can give us more importance on frequent words in a rare document across the entire corpus. This indicates that words common in a document but not in others are considered more significant. This helps machine learning models to perform classification tasks better [1].

Construct models with TF-IDF

```
💡 Click here to ask Blackbox to help you code faster
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import word_tokenize
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Conv1D, MaxPooling1D, Flatten, Embedding, GlobalMaxPooling1D
#from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```
✓ 22.2s

```
tfidf_vectorizer = TfidfVectorizer(tokenizer = word_tokenize, max_features = 300)
tfidf_train = tfidf_vectorizer.fit_transform(x_train)
tfidf_test = tfidf_vectorizer.transform(x_test)
tfidf_combine = np.vstack([tfidf_train.todense(), tfidf_test.todense()])
tfidf_features = tfidf_vectorizer.get_feature_names_out()
```
✓ 9.0s

2.3 Model Training and Evaluation

As for the initial model training phase, we have evaluated the performance of several machine learning models, including Decision Tree, Random Forest, k-nearest Neighbor (kNN), Support Vector Machine (SVM), Naïve Bayes, and Logistic Regression. Each model is designed as the simplest version of the model for the initial evaluation. Then, we would select two target models for further improvement.

2.3.1 Initial Model Performance

Initially, we trained several models for testing, respectively: Decision Tree, Random Forest, kNN, SVC, Naïve Bayes, and Logistic Regression. It turns out a satisfactory accuracy result for each model. The accuracy of each model is respectively 85.21% (Decision Tree),81.27% (Random Forest),80.23%(kNN), 87.63%(SVC), 88.03% (Naïve Bayes) and 88.03% (Logistic Regression). Through our group discussion, we think the model performance is "too ideal" as the initial simple models. In other words, the accuracy of the models might be misleadingly high. It gives us the impression that the model can perform well. It does not accurately identify the minority class (fake news). As a result, we believe there is a certain level of bias towards the majority class (genuine news). Since machine learning algorithms

tend to perform poorly on imbalanced datasets because they are designed to maximize accuracy, the model predicts the majority class (genuine news) for all inputs. This makes the model result in a high accuracy while failing to capture the minority class (fake news) effectively. To solve the accuracy paradox, we decided to re-balance the dataset to minimize the bias within the model.

```
Model Performance Summary:
Model                    Accuracy
------------------------------
Decision Tree:             85.21%
Random Forest:             81.27%
kNN:                       80.23%
SVC:                       87.63%
Naive Bayes:               88.03%
Logistic Regression:       88.03%
```

Figure 1. Test Performance of Initial Base Models

2.3.2 Imbalanced Dataset Handling

Arguably, an imbalanced dataset leads to a biased model. Studies have investigated the effect of the imbalanced dataset on the model's robustness. Kumar et al. proposed that using an uneven training dataset has a significantly negative impact, as insufficient data for predicting each class's characteristics [2]. In our case, our original testing dataset consists of 50000 actual headlines and 11698 fake headlines. Since the number of fake news articles is lower than the genuine ones, it might lead to bias toward the majority class. In other words, the trained machine learning algorithms might perform poorly on imbalanced datasets. To deal with this issue, our group has decided to use random sampling techniques to minimize the bias problem.

Figure 2 shows how we designed the random sampling function to balance the dataset.

```python
# Split n sets of label-balanced data
def get_random_sample_sets(df, n=100, k=5000):
    def find_news(label):
        df_found = df[df["fake"] == label]
        df_sample = df_found.sample(k)
        return df_sample

    def random_samples():
        true_news = find_news(0)
        fake_news = find_news(1)
        concat_df = pd.concat((true_news, fake_news), axis=0)

        # Shuffle only once before splitting
        shuffled_df = concat_df.sample(frac=1)

        return shuffled_df

    random_sample_sets = [random_samples() for _ in range(n)]
    return random_sample_sets
```

Figure 2. Script of the Random Sampling

After creating a random sample set, we attempt to re-evaluate each initial model using a random sample set to see how the improvement is. Through this process, we have found that the test accuracy of the models has dropped by

approximately 10%. The most significant drop in accuracy score among the models is the Decision Tree, which is about 25%. In addition, a confusion map is drawn for each SVM and RF model using different datasets.

We established a confusion map for each model using a population and balanced dataset to evaluate the correctness of the model's prediction on the specific labels. Since we are constructing a fake news detection, we consider Label 0 negative cases and Label 1 positive cases. By this assumption, we define each case as:

- True Positive (TP) for (Ground Truth = 1 & Predicted = 1)
- False Positive (FP) for (Ground Truth = 1 & Predicted = 0)
- True Negative (TF) for (Ground Truth = 0 & Predicted = 0)
- False Positive (FP) for (Ground Truth = 0 & Predicted = 1)

Apart from accuracy, we evaluate each model's performance using 1) Precision, 2) Recall, and 3) F1-score. Precision is computed using TP / (TP + FP), which determines the quality of positive predictions. A high precision indicates that an algorithm provides more relevant results than irrelevant ones. Recall measures the model's capability of detecting positive events precisely with TP / (TP + FN). A high recall score indicates that the classifier correctly predicts the most relevant results. The F1 score is the most critical index for evaluating a model's performance, accounting for precision and recall. A high F1 score generally indicates a well-balanced performance, demonstrating that the model can concurrently attain high precision and recall.

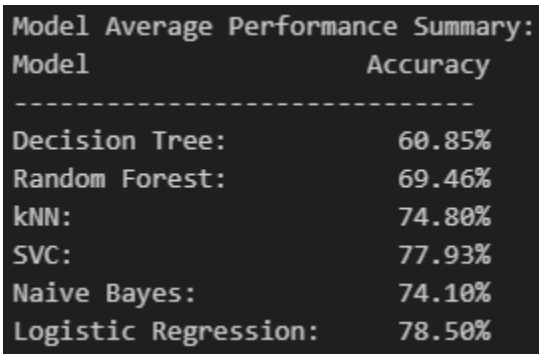


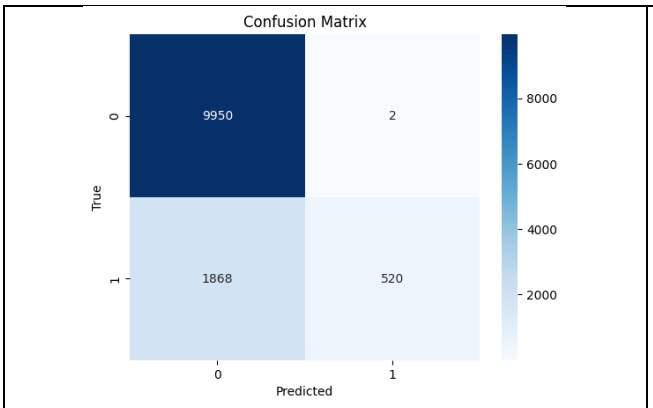
Figure 3. Test Performance of Base Models with Balanced Dataset

	Precision	Recall	F1-Score
Decision Tree	0.2178	0.9962	0.3574
Random Forest	0.0130	1	0.0256
kNN	0.6386	0.4835	0.5503
SVM	0.4108	0.8782	0.5598
Naïve Bayes	0.4209	0.8886	0.5712
Logistic Regression	0.4447	0.8565	0.5854

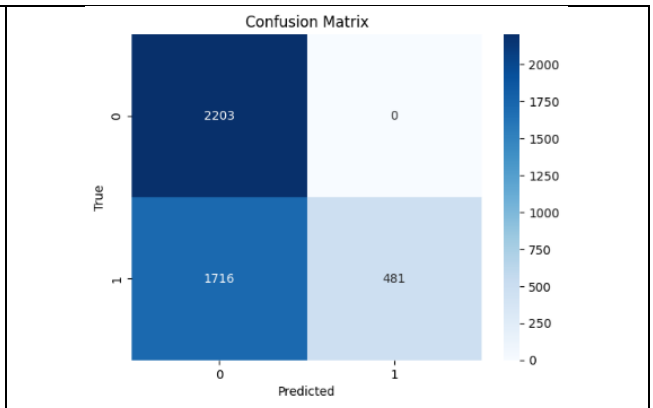
Table 1. Performance of models using population (unbalanced) dataset.

	Precision	Recall	F1-Score
Decision Tree	0.2189	1	0.3592
Random Forest	0.4147	0.9671	0.5804
kNN	0.8138	0.7251	0.7669
SVM	0.8325	0.7637	0.7966
Naïve Bayes	0.8239	0.7692	0.7956
Logistic Regression	0.8298	0.7702	0.7989

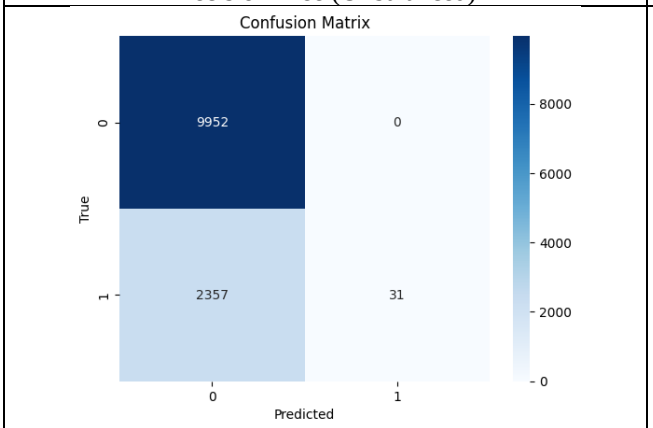
Table 2. Performance of models using a balanced dataset.



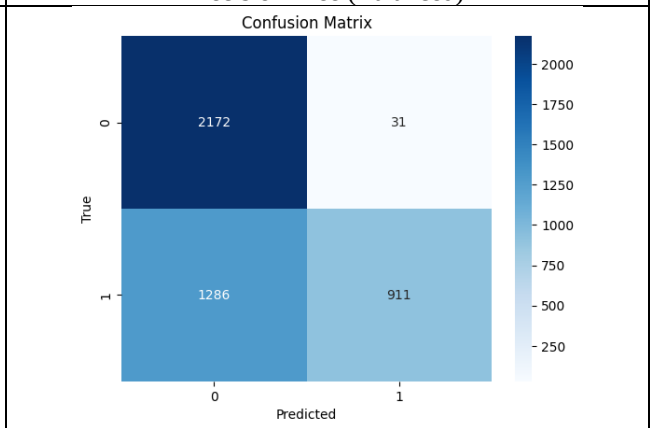
Decision Tree (Unbalanced)



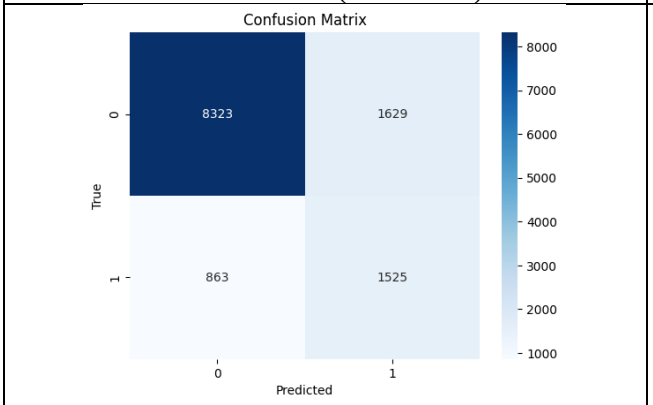
Decision Tree (Balanced)



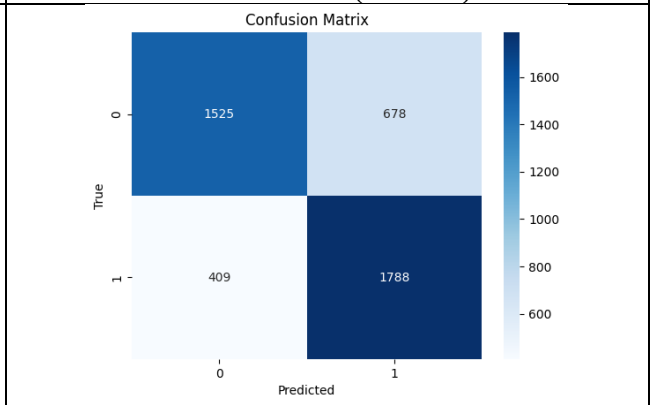
Random Forest (Unbalanced)



Random Forest (Balanced)



kNN (Unbalanced)



kNN (Balanced)

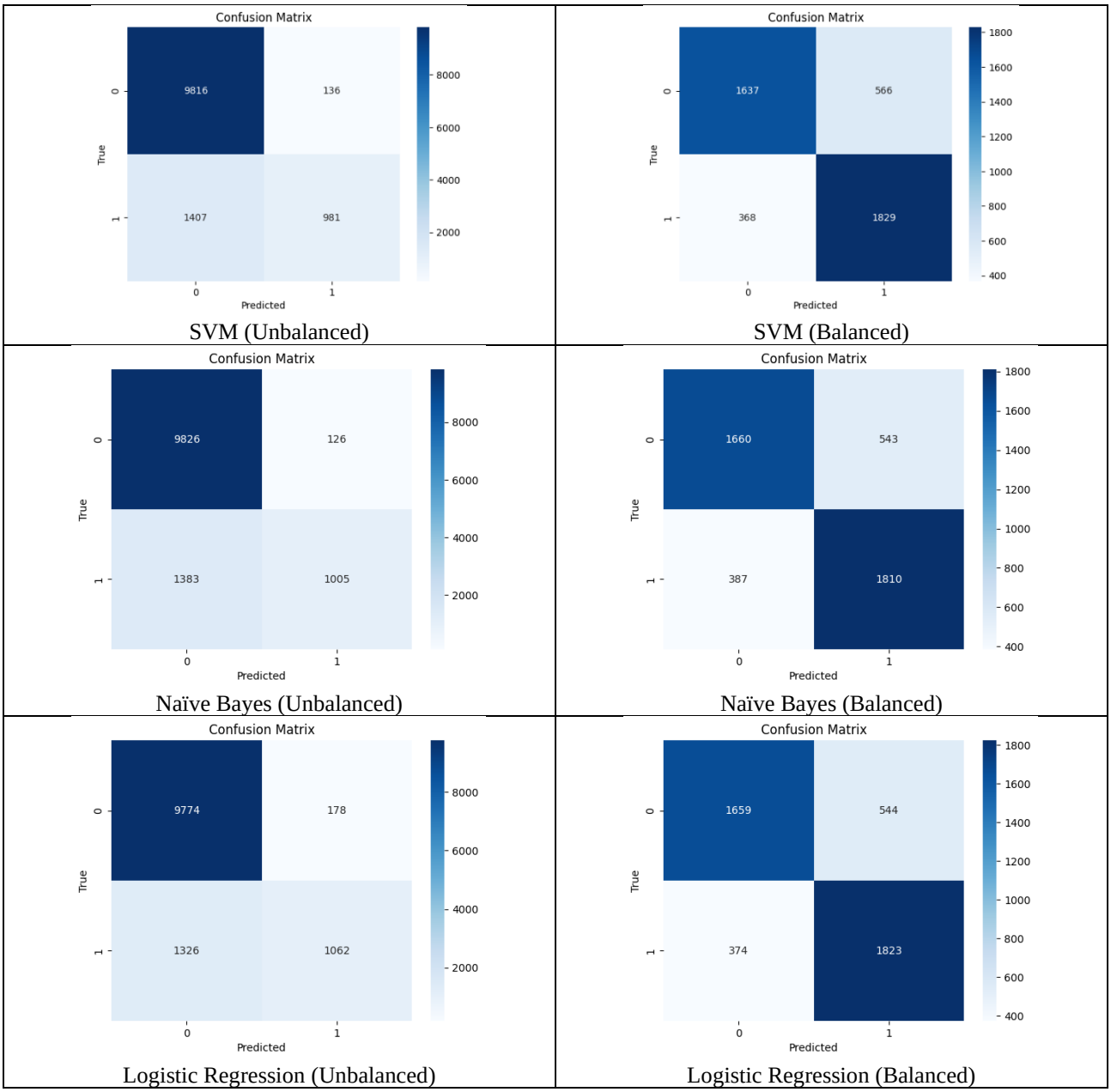


Figure 4. Confusion Map of models using Population (Left) and Balanced Dataset (Right)

The statistics show a significant increase in F1 score from the models using the balanced dataset. The random forest classifier showed the most significant improvement in the F1-score, from 2% to 56%. This indicates that the random forest classifier has considerably improved precision and recall. In addition, the logistic regression classifier achieves the highest F1-score, 79.89%. Overall, it is proven that balancing the dataset improves models' performance, specifically in precision and recall.

2.4 Advanced Techniques for Model Improvement

Our group has selected two models for further improvement targets: the kNN and the Logistic Regression Model. We are choosing the kNN model because it has the potential to improve its performance. We noticed that the initial kNN

model has a descent precision and recall. After using the balanced dataset for training, improvements are found in its precision, recall, and F1 Score. Therefore, we are interested in seeking further enhancement with the kNN model.

On the other hand, we use the logistic regression model as it has the highest F1 score of 79.89% and accuracy of 78.5% trained with the balanced dataset. We are eager to see if any modifications could boost its performance. Hence, in this part, we will demonstrate our techniques to improve the accuracy and model performance.

Base Model Performance in balanced Dataset:

kNN (Precision: 81.38%; Recall: 72.51%; F1-Score:76.69%)

Logistic Regression (Precision: 82.98%; Recall: 77.02% ; F1-Score: 79.89%)

2.4.1 PCA

kNN (Precision: 56.17%; Recall: 81.88% ; F1-Score: 66.63%)

Logistic Regression (Precision: 56.76%; Recall: 84.19% ; F1-Score: 67.8%)

Principal Component Analysis (PCA) is a technique used in machine learning and data science to reduce the dimensions of a dataset. It transforms the original variables into a new set of principal components. These principal components are linear combinations of the original variables and are orthogonal (uncorrelated) to each other. The principal components are ordered based on the variation they retain from the original variables. PCA is beneficial for high-dimensional datasets where the number of features is large compared to the number of observations. By reducing the dimensionality of the data, PCA can help to mitigate the curse of dimensionality, making the data more accessible to visualize and analyze. However, after adopting the PCA technique, neither Knn nor Logistic Regression received noticeable improvement in focusing on the most essential features. This indicates that the effectiveness of PCA in improving model performance depends on the alignment between the principal components it identifies and the features most relevant to the classification task. When PCA does not improve or even degrades model performance, it suggests that the principal components identified by PCA do not capture the essential information for the classification task.

```
# Define the pipeline
pipeline = Pipeline([
    ('pca', PCA(n_components=100)),  # Adjust n_components based on your data
    ('logistic_regression', KNeighborsClassifier(n_neighbors=5, n_jobs = -1))
])
```

2.4.2 Randomized Search

kNN (Precision: 80.35%; Recall: 73.7%; F1-Score: 76.88%)

Logistic Regression (Precision: 82.85%; Recall: 76.21%; F1-Score: 79.39%)

Randomized search is a hyperparameter tuning method that offers a more efficient alternative to Grid Search, especially for complex models with many hyperparameters. Unlike Grid Search, which exhaustively tries all possible combinations of hyperparameters, Randomized Search selects a random subset of the hyperparameter space to explore. This approach significantly reduces the computational cost, making it feasible to tune models with a large number of hyperparameters. However, there is a trade-off: while Randomized Search is faster and less computationally intensive, it does not guarantee finding the optimal combination of hyperparameters. The method relies on statistical sampling to select hyperparameters, which means it might miss the best combination if the sample is not representative enough. Despite this, Randomized Search remains popular for hyperparameter tuning due to its balance between speed and effectiveness. After performing a randomized hyperparameter search on each model, the performance remains unchanged, although we applied the best hyperparameters found during the searches.

2.4.3 Grid Search

kNN (Precision: 80.35%; Recall: 73.7%; F1-Score: 76.88%)

Logistic Regression (Precision: 81.35%; Recall: 77.25%; F1-Score: 79.25%)

Grid Search is a technique used to fine-tune the hyperparameters of a machine-learning model. It works by exhaustively trying all possible combinations of the hyperparameters and systematically exploring a predefined subset of the hyperparameter space. Although computationally expensive, it guarantees finding the best hyperparameters among the tested ones. However, Grid Search may not be practical for models with many hyperparameters or limited computational resources. It is advantageous when the hyperparameter space is not too large and the computational cost is acceptable, as it can lead to significant improvements in model performance by fine-tuning the model's parameters. Similar to the random search, no considerable enhancements are made using the found hyperparameters.

2.4.4 Bagging

kNN (Precision: 84.72% ; Recall: 73.21% ; F1-Score: 78.55%)

Logistic Regression (Precision: 82.16% ; Recall: 77.04% ; F1-Score: 79.52%)

To further enhance the robustness of the model, we are using the ensemble approach. Studies have shown that using bagging models can reduce the variance of the model, therefore improving its overall performance [3]. With prior knowledge, we will use kNN and logistic regression models as the base models for each bagging model. It is noticeable that for each base model, the training data varies from one another even though they are the base models under a bagging model.

After constructing bagging models for each base model type, we evaluate with a sample set extracted from the population. The result shows that the kNN bagging model has an F1 score of 78.55%; meanwhile, the logistic regression bagging model has an F1 score of 79.52%. We noticed that a 3% increment in performance occurred with the kNN model while an insignificant drop with Logistic Regression. Therefore, bagging is a possible approach depending on the base model.

Chapter 3 Conclusion

In this project, we have developed a broad spectrum of models for trial and error. We found out that the number of fake news headlines is lower than the real ones, which leads to a bias toward the majority class. Hence, we have implemented the random sampling technique to address the imbalanced dataset problem.

After extensive research, we concluded that the Logistic Regression model could accurately perform the classification task of detecting fake news, which achieved an impressive precision score of 82.98% after balancing the dataset. However, we have yet to successfully enhance its model performance further during the improvement stage, even though we adopted several techniques. On the other hand, the kNN model performed well after adopting strategies such as PCA, randomized search, grid search, and bagging method. Specifically, after using the ensemble method, the precision value of kNN has significantly improved to 84.72%.

We have learned that achieving mature model performance in machine learning models depends on the dataset's characteristics and enhancing the model architecture for better improvement.

Reference

[1] Singh, A. K., & Shashi, M. (2019). Vectorization of text documents for identifying unifiable news articles. *International Journal of Advanced Computer Science and Applications*, *10*(7). [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2019.0100742

[2] P. Kumar, R. Bhatnagar, K. Gaur, and A. Bhatnagar, "Classification of Imbalanced Data: Review of Methods and Applications," presented at IOP Conference Series Materials Science and Engineering. [Online]. Available: http://dx.doi.org/10.1088/1757-899X/1099/1/012077

[3] Y. Singhai, A. Jain, S. Batra, and Y. Varshney, "Review of Bagging and Boosting Classification Performance on Unbalanced Binary Classification," presented at IEEE 8th International Advance Computing Conference. [Online]. Available: http://dx.doi.org/10.1109/IADCC.2018.8692138